# A Multiqueue Interlacing Peak Scheduling Method Based on Tasks' Classification in Cloud Computing

Liyun Zuo, Shoubin Dong, Lei Shu, *Senior Member, IEEE*, Chunsheng Zhu, *Student Member, IEEE*, and Guangjie Han, *Member, IEEE*

***Abstract*—In cloud computing, resources are dynamic, and the demands placed on the resources allocated to a particular task are diverse. These factors could lead to load imbalances, which affect scheduling efficiency and resource utilization. A scheduling method called interlacing peak is proposed. First, the resource load information, such as CPU, I/O, and memory usage, is periodically collected and updated, and the task information regarding CPU, I/O, and memory is collected. Second, resources are sorted into three queues according to the loads of the CPU, I/O, and memory: CPU intensive, I/O intensive, and memory intensive, according to their demands for resources. Finally, once the tasks have been scheduled, they need to interlace the resource load peak. Some types of tasks need to be matched with the resources whose loads correspond to a lighter types of tasks. In other words, CPU-intensive tasks should be matched with resources with low CPU utilization; I/O-intensive tasks should be matched with resources with shorter I/O wait times; and memory-intensive tasks should be matched with resources that have low memory usage. The effectiveness of this method is proved from the theoretical point of view. It has also been proven to be less complex in regard to time and place. Four experiments were designed to verify the performance of this method. Experiments leverage four metrics: 1) average response time; 2) load balancing; 3) deadline violation rates; and 4) resource utilization. The experimental results show that this method can balance loads and improve the effects of resource allocation and utilization effectively. This is especially true when resources are limited. In this way, many tasks will compete for the same resources. However, this method shows advantage over other similar standard algorithms.**

Manuscript received June 07, 2015; revised January 20, 2016; accepted March 06, 2016. This work was supported in part by the Natural Science Foundation of Guangdong Province, China under Project no. 2014A030313729, in part by 2013 top Level Talents Project in "Sailing Plan" of Guangdong Province, in part by 2014 Guangdong Province Outstanding Young Professor Project, in part by the Science and Technology Key Project of Guangdong under Grant 2014B010112006, and in part by Natural Science Fund of Guangdong under Grant 2015A030308017. *(Corresponding authors: Shoubin Dong and Lei Shu.)*

L. Zuo is with the School of Computer Science and Engineering, South China University of Technology, Guangzhou 510006, China, and also with Guangdong Provincial Key Laboratory of Petrochemical Equipment Fault Diagnosis, Guangdong University of Petrochemical Technology, Maoming 525000, China (e-mail: liyun.zuo@lab.gdupt.edu.cn).

S. Dong is with the School of Computer Science and Engineering, South China University of Technology, Guangzhou 510006, China (e-mail: sbdong@scut.edu.cn).

L. Shu is with the Guangdong Provincial Key Laboratory of Petrochemical Equipment Fault Diagnosis, Guangdong University of Petrochemical Technology, Maoming 525000, China (e-mail: lei.shu@lab.gdupt.edu.cn).

C. Zhu is with the Department of Electrical and Computer Engineering, University of British Columbia, Vancouver, BC V6T 1Z4, Canada (e-mail: cszhu@ece.ubc.ca).

G. Han is with the Department of Information and Communication Systems, Hohai University, Changzhou 213022, China (e-mail: hanguangjie@gmail.com).

Digital Object Identifier 10.1109/JSYST.2016.2542251

***Index Terms*—Cloud computing, load balancing, multiqueue, task classification, task scheduling.**

## I. Introduction

CLOUD computing involves a diverse range of application tasks [1]. Their demands for resources differ accordingly; some require large amounts of storage, and some require CPUs with a powerful computing capacity; others are data intensive and they have considerable I/O. These all cause load imbalance. For example, real-time temperature measurements, and bank deposits and withdrawals involve greater CPU demands and need immediate responses. For these application tasks, the task requests increase sharply as the amount of user access increases. The system's processing speed will be slow, and it may crash, unable to continue service. Additionally, some application tasks require reading and storing data from a database, requiring frequent disk reading and writing to the server, i.e., they require a lot of I/O.

However, there are many dynamic and uncertain factors related to resources and load, such as the dynamically changes in resource nodes over time. Requests for resources also change over the years, seasons, and holidays. Apart from load, resources themselves also undergo many changes, e.g., some resources perhaps join or leave at any time. These dynamic and uncertain factors can lead to a series of problems. If there are too many resources and not enough load, then resources are wasted, but if there is more load than resource capacity, then the performance is affected. This can lead to load imbalance and affect user satisfaction and resource utilization.

For these reasons, to maximize the diversity in user requests and the dynamic factors of resources, not only scheduling methods are required to meet the needs of the response time but also the load must be balanced for each resource in cloud computing.

Most methods balance loads in cloud computing by moving virtual machines, but this can involve high overhead costs [2]–[7]. As such, it is preferable to consider load balancing when scheduling tasks. However, existing studies on scheduling methods usually only consider resources by moving virtual resources [2]–[8] or only consider tasks by optimizing the scheduling algorithm in cloud computing [9]–[14]. The diversity in tasks and the dynamic factors of resources limit effectiveness if they only leverage a single optimization method for allocation in cloud computing. For this reason, it is necessary to assess the scheduling method aimed at task diversity and the dynamic factors of resources to facilitate load balancing and improve system performance.

This paper proposes a multiqueue interlacing peak scheduling method (MIPSM), here called MIPSM. This method is based on the diversity of tasks and the dynamic factors of resources. First, it collects resource and task information, which includes the loads for the CPU, I/O, and memory of resources, and the remands for the CPU, I/O, and memory of tasks. Second, it sorts resources into three queues according to the loads of the CPU, I/O, and memory from small to large. At this time, it also divides tasks into three queues: 1) CPU intensive; 2) I/O intensive; and 3) memory intensive, according to their remands for the CPU, I/O, and memory. Last, it interlaces the resource usage peak when scheduling. Some types of tasks should be scheduled to resources with lighter loads. This could balance loads by making full use of all the idle resources by interlacing the peak of resource usages.

The contributions of this paper are summarized as follows.

1) First, this work proposes a framework model to manage tasks and resources. The framework model includes the task manager, the resource manager, and the scheduler. The task manager can manage task requests and classify tasks. The resource manager is used to collect the information of resource loads and sort resources according to their loads. The scheduler allocates tasks to resources through MIPSM.
2) Second, it proposes a method of task classification. This method is very simple. It only needs some simple mathematical calculation through the existing parameters and does not need to cluster or make extra predictions.
3) Finally, the paper proposes an interlacing peak scheduling method. It establishes three task queues and three resource queues according to the load size for the CPU, I/O, and memory, from small to large. It interlaces the usage peak of resources when scheduling. Some types of tasks, such as those relating to the CPU, should be scheduled to resources whose type load (such as CPU) is lighter, to balance the load and improve allocation efficiency and resource utilization.

This paper is organized as follows. Related work is introduced in Section II. The system model, description of the problem, and queues of tasks and resources are shown in Section III. The interlacing peak scheduling method is presented in Section IV. Complexity and optimization analysis are performed in Section V. Experiments are shown in Section VI, and this paper is concluded in Section VII.

## II. Related Work

Related work includes methods of task scheduling and classification. Studies of task scheduling are typically divided into three categories according to scheduling targets in cloud computing. The first category consists of scheduling methods based on time, including the response times, the best time span, and the completion time. The second is based on performance, such as load balancing and resource utilization. The third is multiobjective optimization, which includes the economic cost, QoS, and energy consumption, with the exception of the above targets.

### A. Methods of Task Classification

There have been a vast number of studies about task diversity. Any of these may help improve task scheduling efficiency and resource utilization. Primarily, they leverage some methods by analyzing the characteristics of tasks and resources or by task classification. For example, one previous work divided the workload into distinct task types with similar characteristics of requirements for resources and performance using the K-means clustering algorithm [15]. Another classified user tasks by QoS preferences, and established the general expectation functions in accordance with task classifications to restrain the fairness of the resources during the selection process [16]. A third focused on analyzing completion times and resource usage, such as CPU, memory, and disk [17]. Its main target was to enhance system performance, including the wait times for task and resource utilization by historical information. Another study analyzed task characteristics and established a model to simulate resource usage patterns and predict resources to optimize resource usage [18]. To plan capacity and schedule tasks, another study assessed the workload and resource consumption (CPU and memory usage) [19]. It divided tasks into small, large, medium, and combinations of these. It classified tasks using the $k$-mean clustering method, which is also used for the Google computing clusters. However, it is highly complex. The complexity of clustering by k-means renders the system nonlinear. Locating the center of the cluster can affect the accuracy of task classification. The previous paper did not mention how to allocate tasks after task classification. Another paper focused on the prediction of task characterization [20]. That paper only considered CPU utilization and did not take memory or I/O into account. Yet another paper considered several systems of task characterization for allocation, distinguishing between CPU- and memory-intensive job, large and small jobs, and so on [21]. However, that paper did not propose any new allocation methods. It did propose a new scheduling structure capable of mixing together some scheduling methods such as randomized first fit and Google algorithm; thus, tasks can select suitable scheduling methods by task characterization.

### B. Focus on Time

This type of research regards the scheduling time as the main target. These studies primarily optimize scheduling algorithms to reduce the associated time allocation using heuristic algorithms and intelligent optimization algorithms, such as in four previous studies [9], [12], [22], [23]. The first of these studies involved a super-heuristic algorithm whose target was to achieve optimal span [9]. The second proposed an adaptive scheduling algorithm on the Hadoop platform whose main target was to reduce completion time [12]. The third proposed two scheduling algorithms from three artificial neural networks (ANNs) and RBF neural networks based on the direct search algorithm optimization (DSO) whose main target was also to find the optimal span [22]. In order to meet the needs associated with high performance and fast scheduling, the fourth study proposed two scheduling algorithms to select the earliest completion of each according to task priority

and to minimize the total execution time of the main target's critical path tasks [23].

### C. Focus on System Performance

This type of research primarily improves load balancing and resource utilization, such as in nine previous studies [6], [7], [24]–[30]. One of these studies proposed a two-stage centralized load balancing framework that had good scalability and high availability and a dynamic algorithm for balancing loads based on neural networks [24]. This assigns weights to each virtual machine (VM) according to the load indicators and indicators of neural networks, dynamically adjusting each VM weight to meet the service level agreement (SLA). Another paper proposed a load-balancing algorithm to balance VM loads and ensure quality of service (QoS) [25]. It reduced the number of VM migrations and migration times during task execution. Another paper took the network performance and load balancing into account, allocating tasks to the nodes of the same types, with minimal transmission link delays [26]. The policy was combined with the first come first served (FCFS), the Min–Min, and the Min–Max algorithms. Another paper balanced load across nodes and reduced the average completion time using the genetic algorithm [27]. Another proposed an improved Min–Min algorithm based on user priorities to achieve load balancing, achieve the best span, and improve resource utilization [28]. It is assumed that users may choose different levels of service according to their needs. The costs corresponding to levels are also different, divided among VIP users and resources to schedule preferentially. But this method is very complex.

### D. Multiobjective Optimization Scheduling

Many research has been devoted to multiobjective optimization, e.g., the constraints of QoS [14], [31]–[33], energy consumption [34], [35], [44], economic costs [3], [7], [14], system performance [8], [11], [13], [37], [38], and all these comprehensive [14], [29], [32], [33], [36], [45].

One previous work proposed a method of dividing resources and budget, minimizing the completion time of tasks, and improving resource utilization [13]. This method takes into account the status of resources. Most other papers focused solely on task optimization and did not considering resource status. Another paper proposed a replication algorithm of the earliest completion time based on task copy [37]. First, it pretreats resources through fuzzy clustering and then schedules tasks using a directed acyclic graph and task duplication. Multi-input multioutput feedback control of a dynamic resource scheduling algorithm was proposed to guarantee optimal effectiveness under time constraints [38]. It considered task execution times, cost, and utilization of resources (CPU, memory).

There are some problems in the existing research working through the above analysis. First, the methods described above require communication, monitoring, and control, all of which produce a certain delay. The complexity is relatively high, and high complexity has negative impacts. Second, some methods classify tasks by clustering. This also increases complexity. Third, some methods move VM, but movement costs may be high. Fourth, many methods of load balancing are based on load prediction (predicting VM requests), but the diversity of the task load and the diversity and heterogeneity in the cloud's hardware environment make it difficult to predict VM requests. Finally, many methods simply optimize the scheduling method itself without considering the dynamic demands of the task for resources or differences in resource loads, with the exception of the literature considering resource status [13].

TABLE I
MAIN NOTATION DEFINITIONS

| Symbol | Definitions |
|---|---|
| $U_i$ | Resource $i$,$1 \leq i \leq N$ |
| $T_j$ | Task $j$,$1 \leq j \leq K$ |
| $N, K$ | The amount of resources and tasks |
| $C_i, O_i, M_i$ | CPU, I/O and memory of $U_i$ |
| $L_j$ | Size of task $T_j$ |
| $D_j$ | The deadline of task $T_j$ |
| $C_j, O_j, M_j$ | CPU, I/O and memory of $T_j$ |
| $C_s, O_s, M_s$ | CPU, I/O and memory of the system |
| $T_{jh}$ | The category of task $T_j$ |
| $C, O, M$ | The rate of $C_j, O_j, M_j$ and $C_s, O_s, M_s$ |
| $Q_C, Q_O, Q_M$ | Resource queues of CPU, I/O and memory |
| $Q_{TC}, Q_{TO}, Q_{TM}$ | Task queues of CPU, I/O and memory |

This paper proposes a scheduling method. First, it not only considers task diversity and reflects differences in demand for resources but also considers the load state of the resource. Second, it also needs to monitor the resource load but only does so periodically and the monitoring process can be done with tasks classification at the same time. In this way, the monitoring process does not take up extra time at any point in the process. Third, this method is relatively simple; it does not need to move VMs or predict the load or runtime. This method is easy to implement and does not involve differential or integral calculation. It is not very complex.

There are three differences between this and other methods. The first is the method of task classification. The method proposed here divides tasks into three queues according to resource demand. It is very simple and easy to implement. The second is that it sorts resources into three resource queues according to resource load states. Third, it selects resources by staggering the usage peak during allocation. This method assigns certain types of tasks to the same types of resources, the idlest. Thus, this method can achieve effective load balancing. However, it can assign three tasks, one from each of each of these three queues, to different resources to perform at the same time. This could significantly improve allocation efficiency.

## III. SYSTEM MODEL

This section describes the system framework, the definitions of tasks and resources, and the queues of resources and tasks. The primary parameters and their meanings are listed in Table I.

### A. System Framework

In this paper, the system framework model is shown in Fig. 1, in which the task manager manages task requests that the user submits. At the same time, it analyzes and processes these requests and divides tasks into three queues: 1) CPU intensive;

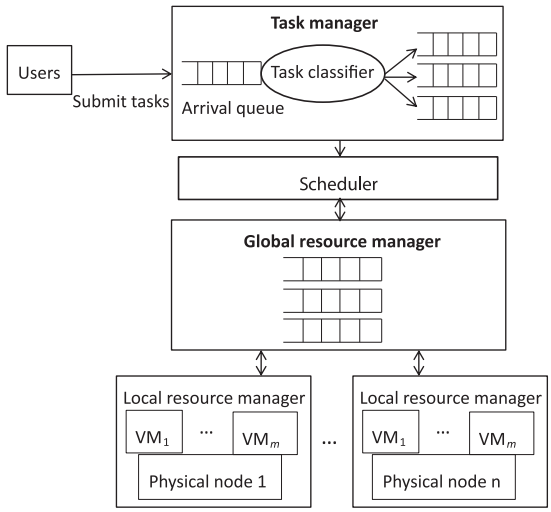



Fig. 1. System framework model of the scheduler, task and resource manager.

2) I/O intensive; and 3) memory intensive, according to their demands for resources.

The local resource manager monitors and manages local resource nodes. It periodically monitors local virtual resources to determine their CPU, I/O, and memory load information, and to submit the information to the global resource manager. The global resource manager periodically collects and updates information from local resource managers. It then divides resources into three queues according to the loads of the CPU, I/O, and memory from small to large.

The scheduler collects this task and resource information from the task manager and the global resource manager. The scheduler is responsible for allocating tasks to resources using the interlacing peak scheduling method, which this paper proposes, according to this information regarding the tasks and resources.

### B. Definitions of Resources and Tasks

First, it is assumed that there are $N$ resources $U = \{u_1, u_2, \ldots, u_i, \ldots, u_N\}$ and $K$ tasks $\{T_1, T_2, \ldots, T_j, \ldots, T_K\}$ in the current system of cloud computing. Here, cloud resources refer to virtual resources.

*Definition 1: Resources:* Here, a resource is a single virtual machine. Each virtual resource cloud is defined by the parameters of its CPU, I/O, and memory. That is to say, $U_i = (C_i, O_i, M_i)$. These three parameters are representative of CPU utilization, I/O waiting time, and memory usage. These parameters come from the global resource manager, which periodically collects and updates information from local resource managers.

*Definition 2:* $Tasks, T_j = (C_j, L_j, M_j, D_j)$: The first three parameters are 1) CPU usage; 2) task size; and 3) memory that the user applies to use, and $D_j$ is the deadline of the task. Task size is the size of the task, and it is equal to the length of the task. These parameters come from the task manager and are submitted by users.

It is also necessary to determine I/O usage information, excepting requests for CPU and memory. Definition 3 calculates the parameter of I/O usage as follows.

*Definition 3:* The I/O usage of the task $T_j$ is defined as $O_j = \frac{L_j}{C_j}$. $L_j$ is the task size, and $C_j$ shows the capacity of CPU completing the task $T_j$. The I/O usage of the task $T_j$ cannot be determined directly. I/O usage is concerned to the task size and the capacity of CPU. Here, I/O usage is estimated by the rate of $L_j$ and $C_j$.

*Definition 4:* According to these definitions, task resource requirements and resource capacity in the system were defined as follows. These parameters $(C_i, O_i, M_i)$ reflect the resource capacity which can provide the CPU, I/O, and memory. At the same time, these parameters $(C_j, O_j, M_j)$ reflect the tasks resource requirement of the CPU, I/O, and memory.

*Assumption 1:* It is assumed that the information of user submitted is trusted, that the information regarding resource demand, which is submitted by the user, is accurate where the parameter $L_j$ can be provided accurately. $D_j$ is the deadline by which the user expects that the task will be completed. The parameters $(C_j, M_j)$ are the values estimated by the user and are enough to complete the task.

*Assumption 2:* The parameters of the task may change during its lifetime; most tasks have their phase behaviors. This means that a task could be CPU intensive during one stage but memory intensive during another stage. In this paper, for simplicity, it is here assumed that these parameters are fixed and do not change during the lifetime of the task.

*Assumption 3:* The resources are dynamic in cloud computing. Therefore, these parameters $(C_i, O_i, M_i)$ are the original value. According to the system model, the local resource manager periodically monitors local virtual resources to determine their CPU, I/O, and memory load information and to submit the information to the global resource manager. The global resource manager periodically collects and updates information from local resource managers. This means that these parameters $(C_i, O_i, M_i)$ are updated periodically.

In cloud computing, virtualization technology can be used to monitor resource usage. If the capacity of the resources that the user is accessing, such as CPU and memory, exceeds that requested by all users during actual implementation, the system will cut off the task performance. This means that the task fails. This shows that assumption 1 is reasonable.

## IV. Interlacing Peak Scheduling Method Based on Multiqueue

This section introduces the MIPSM, based on task and resource queues. It includes three parts: 1) task classification; 2) resource sorting; and 3) interlacing peak scheduling.

### A. Tasks Classification

First, the task manager manages user-submitted task requests. The task request information includes CPU usage, task size, and memory in order to reflect the diversity of tasks needed to collect the necessary information that tasks demand for resources. Consequently, task classification needs information regarding I/O usage. The I/O usage is calculated using the task size and CPU. In this way, it collects all the information for the CPU, I/O, and memory.

Second, it is necessary to determine the values of $C_s$, $O_s$, and $M_s$ of CPU, I/O, and memory in the system before dividing the tasks. Then, task classification must calculate the ratios of each task of CPU, I/O, and memory in the system. For each task $T_j$, calculate these ratios $C$, $O$, and $M$ by its parameters $C_j$, $O_j$, and $M_j$ and $C_s$, $O_s$, and $M_s$. The largest of these three ratios is regarded as the task category $T_{jh}$

$$T_{jh} = \max(C, O, M) = \max\left(\frac{C_j}{C_s}, \frac{O_j}{O_s}, \frac{M_j}{M_s}\right). \quad (1)$$

Finally, $K$ tasks are divided into three queues $Q_{TC}$, $Q_{TO}$, and $Q_{TM}$ of $a$ CPU intensive, $b$ I/O intensive, and $K-a-b$ mem intensive by the task category $T_{jh}$ as follows:

$$Q_{TC} = \{T_1, T_2, \ldots, T_{jC}, \ldots, T_a\} \quad (2)$$

$$Q_{TO} = \{T_{a+1}, T_{a+2}, \ldots, T_{jO}, \ldots, T_{a+b}\} \quad (3)$$

$$Q_{TM} = \{T_{a+b+1}, T_{a+b+2}, \ldots, T_{jM}, \ldots, T_{K-a-b}\}. \quad (4)$$

For example, if $T_{jh} = C$, then the task $T_j$ will be divided into the queue of CPU intensive. If $T_{jh} = O$, then the task $T_j$ will be divided into the queue of I/O intensive.

Here, each one of the three queues makes up only one portion of all tasks. The total of these three queues equals the number of tasks.

### B. Resource Sorting

First, the local resource manager collects information regarding CPU utilization, I/O wait times, and memory usage from local virtual machines. Next, it submits the information to the global resource manager. Third, the global resource manager sorts all resources according to these three parameters from small to large, respectively, forming three queues $Q_C$, $Q_O$, and $Q_M$ as follows:

$$Q_C = \{U_1, U_2, \ldots, U_{iC}, \ldots, U_N\} \quad (5)$$

$$Q_O = \{U_1, U_2, \ldots, U_{iO}, \ldots, U_N\} \quad (6)$$

$$Q_M = \{U_1, U_2, \ldots, U_{iM}, \ldots, U_N\}. \quad (7)$$

Here, all the resources sort rather than classify, due to the amount and dynamism of the resources. Therefore, each of the three queues includes all resources, unlike the task queues.

### C. Interlacing Peak Scheduling Method

The interlacing peak scheduling method, as the name suggests, interlaces the resource usage peaks when scheduling tasks to resources.

The principle of this method is that the CPU, I/O, and memory usages do not conflict with each other. For example, some task requests for big data computing need only a small quantity of I/O resources; however, they require significant CPU resources to compute and process. This is so it can effectively achieve load balancing because it makes full use of all idle resources by interlacing the peak resource usage.

First, the task manager manages task requests submitted by users. The task request information includes CPU usage, task size, and memory. Thus, it obtains all the information from the CPU, I/O, and memory. It then divides tasks into three queues that are 1) CPU intensive; 2) I/O intensive; and 3) memory intensive, according to these three parameters.

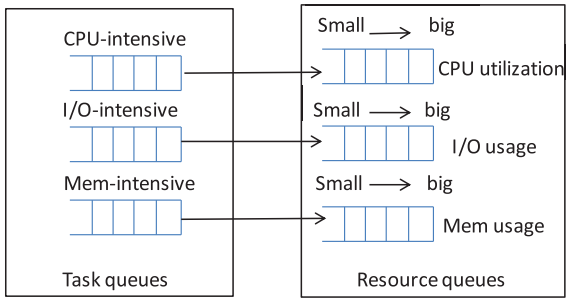



Fig. 2. Interlacing peak scheduling method based on task and resource queues.

Second, the interlacing peak scheduling method needs to collect the resource load information, such as the CPU, I/O, and memory usage. For this reason, the local resource manager monitors the local virtual resource loads to collect their CPU, I/O, and memory load information. Then, the local resource manager submits the information to the global resource manager, which periodically collects and updates the CPU, I/O, and memory load information from local resource managers. At the same time, the global resource manager sorts the resources into three queues according to the CPU, I/O, and memory loads from small to large. The resource manager only sorts, so each of three queues includes all the resources. This is different from the task queues.

Finally, the scheduler allocates tasks to resources using the interlacing peak scheduling method. This method is based on resource sorting and task classification. It interlaces the resource usage peaks when scheduling. Some types of tasks should be scheduled to the resources whose load (the type of load is corresponding to the task type) is lighter. The CPU intensive tasks are scheduled to those resources with low CPU utilization; the I/O intensive tasks are scheduled to those resources with short I/O wait times, and the memory intensive tasks are scheduled for resources with low memory usage. The details of the process are shown in Fig. 2.

Fig. 2 depicts the scheduler process allocating tasks to resources. There are two kinds of queues: 1) the task queue and 2) the resource queue. The task queue includes three queues: 1) CPU intensive; 2) I/O intensive; and 3) memory intensive. The resource queue also has three queues. They are formed by sorting the CPU, I/O, and memory loads from small to large. The scheduler then allocates the tasks to the resources.

The implementation of MIPSM is shown as the pseudocode of Algorithm 1.

### D. Effectiveness of the Algorithm

As shown in the description of Section IV-B, MIPSM staggers the resource usage peaks when scheduling. It assigns certain types of tasks to the types of resources that are the idlest. In this way, it prevents resources from becoming overloaded or too idle, so it balances the load effectively. However, MIPSM

**Algorithm 1** Multiqueue Interlacing Peak Scheduling

**Input:**
$T_1, T_2, \ldots, T_j, \ldots, T_K, T_j = (C_j, L_j, M_j, D_j), Q_C, Q_O, Q_M, C_j, O_j, M_j, C_s, O_s, M_s$
**Output:**
$(T_j, Q_x)$
1: BEGIN
2: FOR j=1 to K
3: Calculate $O_j$ using Definition 3;
4: Calculate $T_{jh}$ using Formula 1;
5: IF $T_{jh} = C$ Then
6: x=C;
7: $T_j \rightarrow Q_C$;
8: END IF
9: IF $T_{jh} = O$ Then
10: x=O;
11: $T_j \rightarrow Q_O$;
12: END IF
13: IF $T_{jh} = M$ Then
14: x=M;
15: $T_j \rightarrow Q_M$;
16: END IF
17: END FOR
18: END

divides tasks into three queues; it can assign three tasks, one from each of the three queues, to different resources so that they are performed at the same time. This method can greatly improve allocation efficiency. This method's effectiveness in load balancing is proven below.

It is assumed that in the cloud computing system, there are $N$ resources $U = \{u_1, u_2, \ldots, u_i, \ldots, u_N\}$ and $K$ tasks $\{T_1, T_2, \ldots, T_j, \ldots, T_K\}$, three resource queues $Q_C$, $Q_O$, and $Q_M$, and a task set including three queues $Q_{TC}$, $Q_{TO}$, and $Q_{TM}$ of $a$ CPU intensive, $b$ I/O intensive, and $K - a - b$ mem intensive.

*Proof:* The interlacing peak scheduling method can make the system resources load balance according to MIPSM. ■

*Proof:* 1) When $N \geq K$, tasks are assigned according to Algorithm 1 as follows:

$$Q_{TC} \rightarrow \{U_1, U_2, \ldots, U_a\} \tag{8}$$
$$Q_{TO} \rightarrow \{U_1, U_2, \ldots, U_{a+b}\} \tag{9}$$
$$Q_{TM} \rightarrow \{U_1, U_2, \ldots, U_{k-a-b}\}. \tag{10}$$

This ensures that tasks are assigned to idle resources, and the system's CPU utilization, I/O wait time, and memory roughly are same to maintain load balancing.

2) When $N < K$, the number of resources is less than the number of tasks, so the tasks need to be assigned in batches.

First, the front $h (h = \lceil \frac{N}{3} \rceil)$ tasks are selected in the queue assigned by Algorithm 1 as follows:

$$Q_{TC} \rightarrow \{U_1, U_2, \ldots, U_h\} \tag{11}$$
$$Q_{TO} \rightarrow \{U_1, U_2, \ldots, U_h\} \tag{12}$$
$$Q_{TM} \rightarrow \{U_1, U_2, \ldots, U_h\}. \tag{13}$$

Second, the remaining $K - h$ tasks will be assigned to the next batch. If $K - h < N$, then the $\lceil \frac{K-h}{3} \rceil$ tasks will be assigned to the three resource queues; otherwise, it takes the front $\lceil \frac{N}{3} \rceil$ tasks to assign again, and the remaining $K - 2h$ tasks will be assigned to the next batch.

In this case, each resource can receive at least one task, and the amounts of CPU utilization, I/O waiting time, and memory usage are equal. The system resources keep the load balanced.

Quod Erat Demonstrandum (QED).

## V. Complexity and Optimization Analysis of MIPSM

As indicated in the description given in Section IV, MIPSM divides tasks into three task queues: $Q_{TC}$, $Q_{TO}$, and $Q_{TM}$, according to the tasks demand for resources. At the same time, it sorts resources to form three resource queues: $Q_C$, $Q_O$, and $Q_M$, according to the load state of resources. Finally, MIPSM staggers the peak resource usage when scheduling and assigns certain tasks to the resources that are the idlest. In this way, it can balance load effectively, which was proved in Section IV. It can also assign three tasks, on from each of the three queues, to different resources and set them to perform at the same time. This can significantly improve scheduling efficiency. This section describes the analysis of the optimization effects and complexity of MIPSM.

### A. Complexity of Algorithm

The complexity of the scheduling algorithm may have some effect on the system. MIPSM includes three parts: 1) task classification; 2) resource sorting; and 3) scheduling; the MIPSM complexity analysis is performed from these three parts.

The algorithm's time complexity is related to the number $N$ of resources and the number $K$ of tasks. The task classification time complexity is $O(K)$. The complexity of the sorting resource time is $O(N \lg N)$. Finally, the scheduling time complexity is $O(K)$. Therefore, the total time complexity is $O(N \lg N + K)$. In fact, resource sorting and task classification can occur at the same time, and resources only sort periodically. The actual time complexity is less than $O(N \lg N + K)$.

For space complexity, resource sorting and task classification are both $O(1)$, and the scheduling algorithm is also $O(1)$. For this reason, the total space complexity is $O(1)$.

The scheduling method in this paper is simple and does not involve differential or integral calculations; the time complexity and space complexity are relatively low.

### B. Optimization

According to the principle of MIPSM, three queues of tasks can be executed in different resources at the same time. Ideally, the whole system would have been optimized three times. In a worst-case scenario, there may be three tasks, one from each of the three queues, sent to the same resource at the same time in the original state. In this case, optimization is zero.

TABLE II
VM SETUP OF DATA CENTER

| Parameter | Value |
|---|---|
| CPU computing ability | 1860 MIPs, 2660 MIPs |
| Disk I/O | 10 GB |
| RAM | 4096 MB |
| Bandwidth | 100 M/s |
| Storage | 10 G |

TABLE III
TASK SETUP OF DATA CENTER

| Parameter | Value |
|---|---|
| Length (CPU) | [400,1000] MIPs |
| File size | [200,1000] MB |
| Output size (memory) | [20,40] MB |

In this way, the upper limit of optimization is 3 and the lower limit is 0.

However, the worst case only occurs in the original state. The load balancing of system resources can improve with the implementation of MIPSM. Afterward, this worst-case scenario will not happen again. The optimization limit must be greater than 0.

## VI. EXPERIMENTS

Some experiments were designed using Cloudsim 3.0 to confirm the performance of MIPSM [39]. There were two types of experiments. First, some simulation experiments used the tasks, which were generated by Cloudsim and had clear CPU, memory, and I/O (Sections VI-A–VI-F). Second, in order to demonstrate the effect in real application, some experiments used real-world workloads (Section VI-G) .

### A. Experiment Setup

A data center was simulated using Cloudsim 3.0. It had 100 hosts. There were 10 virtual machines on each host. The setup is as shown in Table II.

Each experiment generated a series of task requests. Each of these task requests has an obvious need for CPU, I/O, and memory. The number of tasks is $K = 500$ $(N > K)$ and $K = 1200$ $(N > K)$ to fully verify the method's performance. Task parameters are as shown in Table III.

There are two basic algorithms Min–Min [40], [41] and least connection (termed LC) [42], to compare with MIPSM. The Min–Min algorithm has better completion time, and the LC algorithm has better load balancing because it prefers idle resources every time.

### B. Experiment Metrics

Experiments use four performance-evaluation indicators. The first is average response time, which is used to evaluate scheduling performance. The second is the load balancing, used to verify resource load balancing by the utilization of randomly selected resources. The third is the deadline violation rate, which is the feedback effect of QoS because a perfect stable system requires feedback to verify its performance. The fourth is the resource utilization.

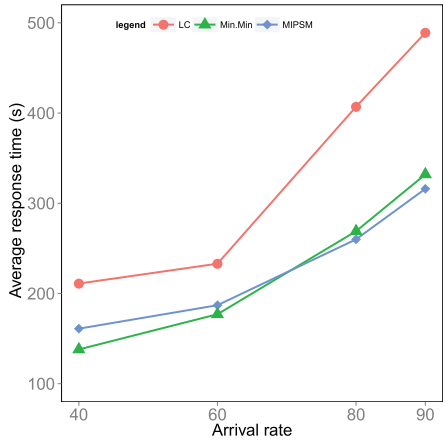



Fig. 3. Average response time with different task arrival rates when $K = 500$ and $K < N$.

The main objective of MIPSM is efficiency and load balancing. Load balancing is reflected through CPU, I/O, and memory resource usage. Scheduling efficiency is reflected through the response times and deadline violation rates. The response time of each task is from the task submitted to return the result. It includes the waiting time $t_{j-\text{wait}}$ and the completion time $t_{j-\text{complete}}$. The average response time $t_{\text{response}}$ is calculated as (14). For the deadline violation rates, if the running time of task $T_j$ is greater than the deadline $D_j$, the task is considered to violate the deadline constraints. The deadline violation rate $v$ is calculated as (15)

$$t_{\text{response}} = \frac{\sum(t_{j-\text{wait}} + t_{j-\text{complete}})}{K} \tag{14}$$

$$v = \frac{n_d}{K} * 100\% \tag{15}$$

where $n_d$ is the number violating the deadline time in $K$ task.

The experiments in real-world workloads' part used three metrics: 1) the average resource utilization of the system; 2) the average response time; and 3) the average slowdown. The average resource utilization of the system was used to show the average utilization of CPU, memory, and I/O. The average response time was used to verify the efficiency of allocation. The slowdown reflected the overhead of these methods indirectly. The slowdown is calculated as

$$\text{slowdown}_j = \frac{t_{j-\text{wait}} + \max(t_{j-\text{complete}}, 10)}{\max(t_{j-\text{complete}}, 10)} \tag{16}$$

where $\max(t_{j-\text{complete}}, 10)$ means if the completion time is less than 10 s, the completion time will be replaced with 10.

### C. Response Time

The first experiment verified task-scheduling performance by response time. The task arrival rates were 40, 60, 80, and 90 tasks/s. The results are shown in Figs. 3 and 4.

Figs. 3 and 4 show that the average response times of all these three algorithms increase as the task arrival rates increase, especially when the arrival rate is 80. MIPSM showed the best performance; the Min–Min algorithm performed better than the

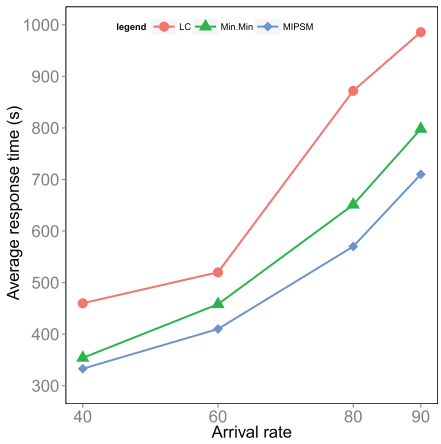



Fig. 4. Average response time with different task arrival rates when $K = 1200$ and $K > N$.

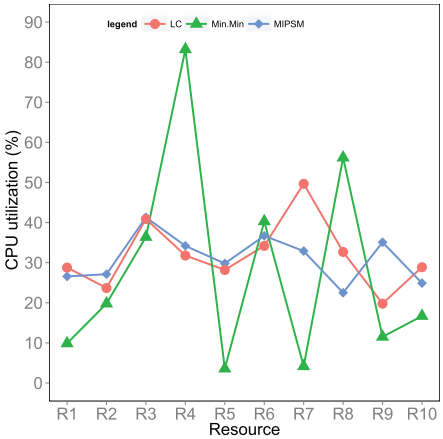



Fig. 5. CPU utilization with different resources when $K = 500$ and $K < N$.

LC algorithm in these three algorithms. At first, MIPSM was less than the Min–Min algorithm, but MIPSM was better than the Min–Min algorithm with increase in the arrival rate; when $K > N$, MIPSM had more obvious advantages. This is because the Min–Min algorithm prioritized small tasks and gave preference to good resources, so it showed better performance when there were fewer tasks. When there are more tasks, MIPSM is better because it considers the scheduling performance and load balancing. In this way, it can keep the load balanced when the quantity of resources is lower than that required by tasks and demonstrates good performance on an average response time.

### D. Load Balancing

The second experiment verifies the performance of task scheduling through load balancing. The experiment selected 10 resources at random and then observed their CPU, I/O, and memory usage. The experiment ran 10 times and average values are shown. The results are shown in Figs. 5–10.

Figs. 5–10 show there to be obvious fluctuations in CPU, I/O, and memory usage for the Min–Min and LC algorithms. However, fluctuations in MIPSM were very slight. This means that MIPSM had the best effect on load balancing. It showed

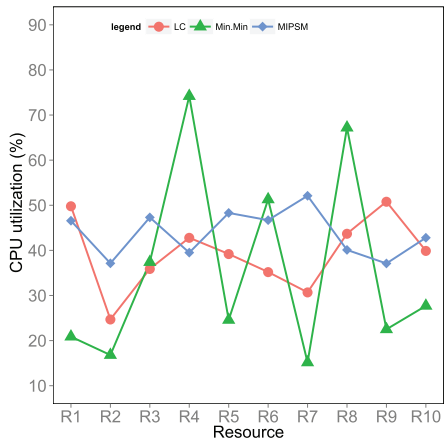



Fig. 6. CPU utilization with different resources when $K = 1200$ and $K > N$.

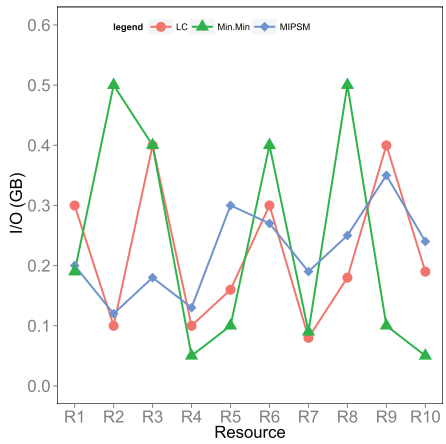



Fig. 7. I/O utilization with different resources when $K = 500$ and $K < N$.

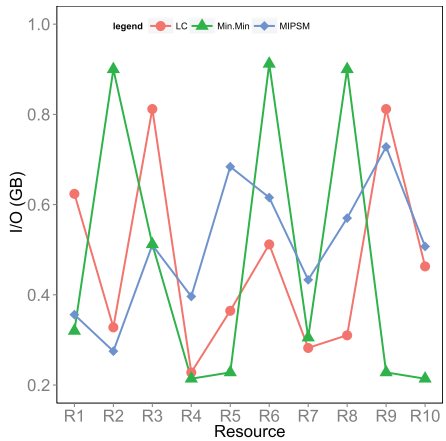



Fig. 8. I/O utilization with different resources when $K = 1200$ and $K > N$.

greater resource utilization than the other two. The Min–Min algorithm showed the worst load balancing, while the LC algorithm always preferred idle resources, so it can balance loads. However, this is still less than MIPSM, especially when the number of tasks is high. The Min–Min algorithm always preferred good resources, which caused considerable load imbalances. For the two cases of $K < N$ and $K > N$, the resource utilization of $K > N$ was higher.

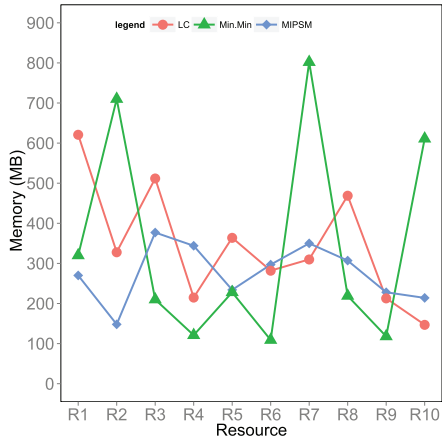



Fig. 9. Memory utilization with different resources when $K = 500$ and $K < N$.

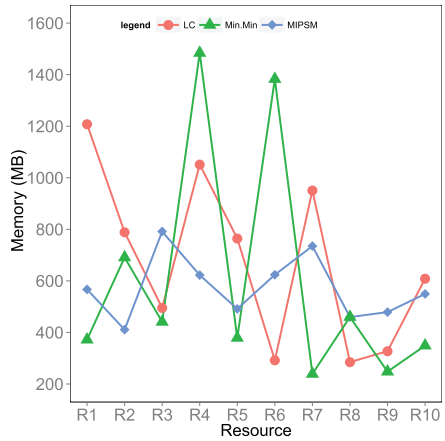



Fig. 10. Memory utilization with different resources when $K = 1200$ and $K > N$.

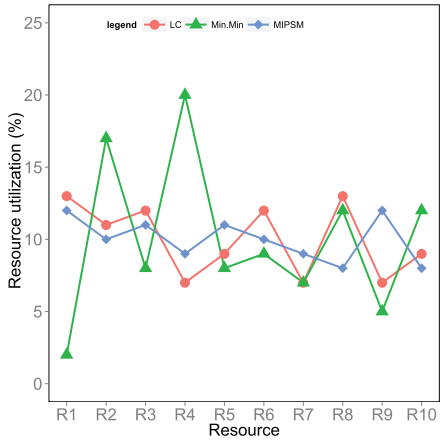



Fig. 11. Resource utilization of 10 resources randomly selected when $K = 500$ and $K < N$.

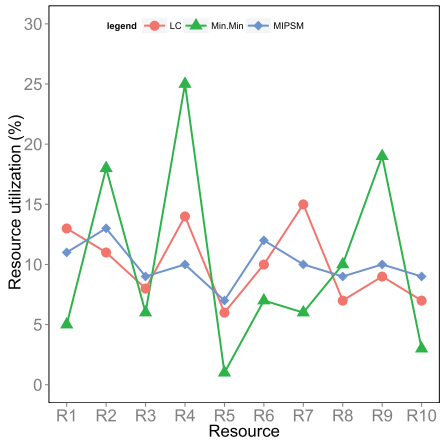



Fig. 12. Resource utilization of 10 resources randomly selected when $K = 1200$ and $K > N$.

### *E. Resource Utilization*

The third experiment was used to verify the scheduling method's resource utilization. First, experiment randomly selected 10 resources to observe the resource usage when $K < N$ and $K > N$; here, the resources included CPU, I/O, and memory. Second, these three algorithms were scheduled twice (once one ended, the next was randomly generated). The task arrivals of each time were 10, 40, 60, 80, and 90. Third, the experiment was performed 10 times. For every algorithm, the scheduling was successfully executed 100 times. For this reason, resource utilization was here defined as the number of times each resource was scheduled. The results are shown as Figs. 11 and 12.

Figs. 11 and 12 show resource utilization. The utilization of ten resources with MIPSM showed little difference. The LC algorithm was similar to MIPSM, meaning that these two algorithms both balanced the load very well. The Min–Min algorithm was the worst. The fluctuation ranges in MIPSM, LC and Min–Min algorithm were 4%, 6%, and 18%, respectively, when $K < N$. However, when $K > N$, there was resource competition. The fluctuation ranges of these three algorithms were 5%, 9%, and 24%, respectively; MIPSM and the LC algorithm still performed better than the Min–Min algorithm. MIPSM was better than the LC algorithm. This showed the advantage of MIPSM with respect to resource utilization and load balancing.

### *F. Violation Rate of Deadline*

The fourth experiment confirmed the scheduling QoS by assessing deadline violation rates. The rates at which deadlines for different task arrival rates were missed are shown in Figs. 13 and 14.

Figs. 13 and 14 show that the deadline violation rate increases as the task arrival rates increase accordingly. MIPSM shows the best performance in three algorithms. The Min–Min algorithm was found to be better than that of the LC algorithm. When $K > N$, resource competition took place, which left the deadline violation rates of three methods relatively large. However, the deadline violation rates of MIPSM are all less than 8%. This is an acceptable limit.

### *G. Experiments by Real-World Workload*

The experiments used real workload, in this case the high-performance computing cluster of Shenzhen the Beijing

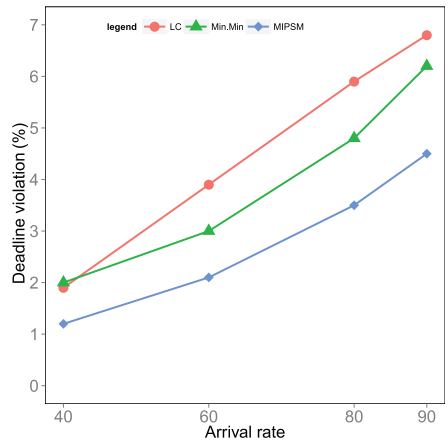



Fig. 13. Violation rate of deadline with different task arrival rates when $K = 500$ and $K < N$.

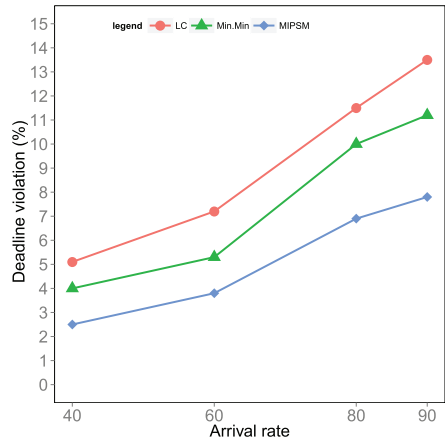



Fig. 14. Violation rate of deadline with different task arrival rates when $K = 1200$ and $K > N$.

Genomics Institute (BGI) [43]. The real workload data were collected from March 5 to March 11, 2012. The workload included 232 339 tasks. These tasks required not only a CPU, but also memory and I/O. The users were required to estimate the resources for their tasks when submitting them to the BGI system. This is consistent with the definition given in Section III.

Some similar methods were compared to MIPSM. First, the method of task classification in one previous paper was combined with the interlacing peak scheduling method [19]. Here, the tasks that required a large CPU were scheduled to the CPU resource queue. The tasks of large memory were assigned to the memory queue, and the tasks whose CPU and memory were medium and small were scheduled to the I/O queue. The experiments also used the FCFS algorithm in the real BGI system and the first fit algorithm in the paper [21].

In order to evaluate the performance with different load, the original arrival time interval of tasks was compressed to produce heavier load data than the original data. Five load data (whose load values were 1, 1.25, 1.5, 1.75, and 2, respectively) were selected. A load value of 1 means the original arrival time interval, 1.25 indicated that the arrival time interval was 1/1.25 times to the original values, a value of 1.5 indicated that the

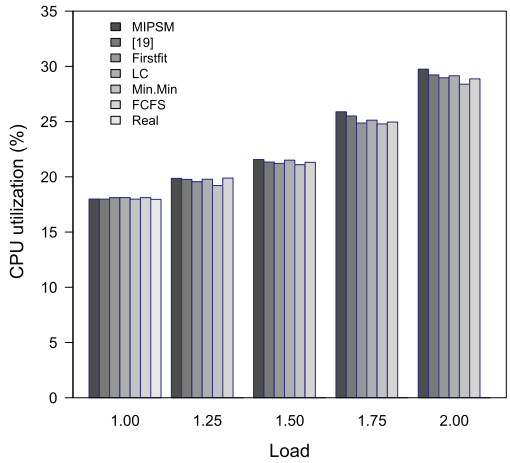



Fig. 15. Average CPU utilization with different load of real-world workload.

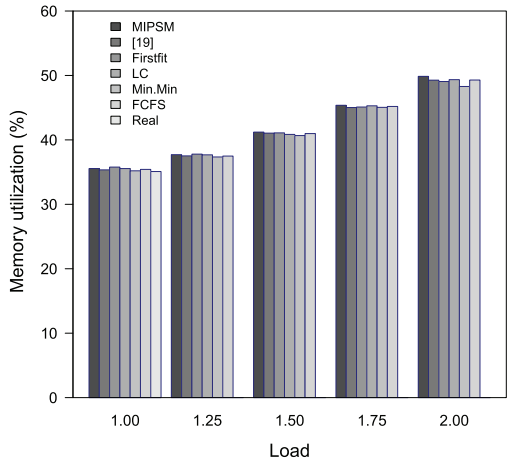



Fig. 16. Average memory utilization with different load of real-world workload.

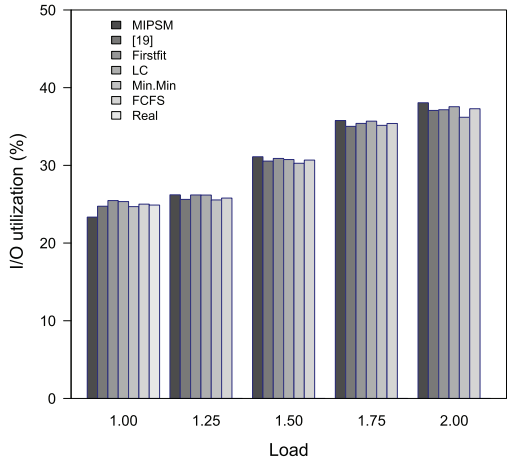



Fig. 17. Average I/O utilization with different load of real-world workload.

arrival time interval was 1/1.5 times, and so on. The results are shown as Figs. 15–19.

In these figures, the term "real" refers to the actual data in the real system and its load value was 1. The results showed that the resource utilization was associated with the load. MIPSM, a previous work, Firstfit, and LC had better performance than

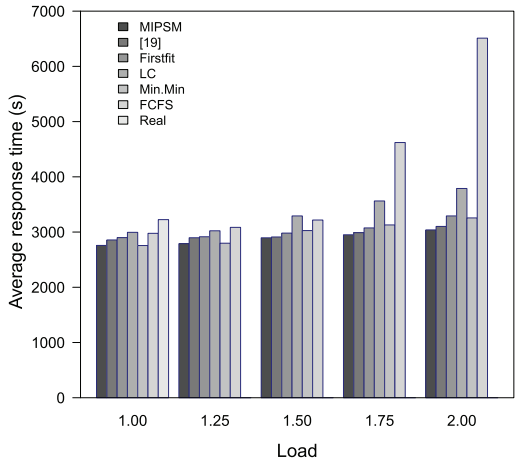



Fig. 18. Average response time with different load of real-world workload.

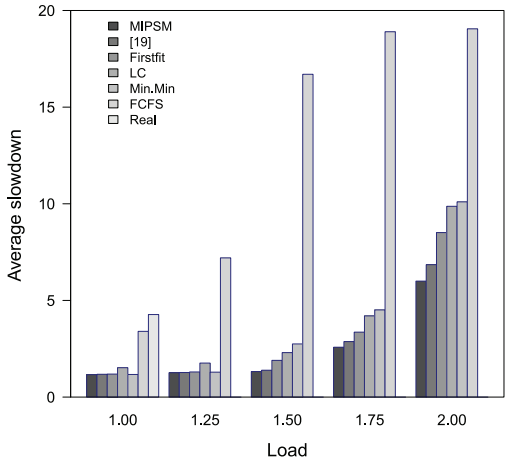



Fig. 19. Average Slowdown with different load of real-world workload.

the real system. For the average response time and slowdown, when the load was low, the difference of all methods was small, and they were all better than the original system and FCFS. However, when the load was high, the average response time and slowdown of MIPSM and of the previous work were superior to Firstfit, LC, Min–Min, and FCFS [19]. The advantage with respect to response time of Min–Min slowly disappeared as the load increased. This proved that task classification and the interlacing peak scheduling method were very effective. Besides, the method described in the previous work was slightly worse than MIPSM [19]. This is because the task classification in that work produced more time overhead when using *k*-means clustering method in the second step [19].

## VII. Conclusion

In cloud computing, resource load dynamics, task diversity, and the differences in resource task demand can lead to load imbalance and affect the efficiency of task scheduling and resource utilization. In order to solve these problems, a MIPSM is here proposed. First, tasks were divided into three queues: 1) CPU intensive; 2) I/O intensive; and 3) memory intensive. Second, the resources were sorted according to CPU utilization loads, I/O wait times, and memory usage. Last, three queues of tasks were scheduled to those resources whose loads for the corresponding metric (CPU, I/O, or memory) were lighter than the others. This method was found to balance loads effectively through theoretical verification. The simulation system was designed to verify the performance through experimentation. MIPSM was compared to the Min–Min algorithm and the LC algorithm because the Min–Min algorithm had advantages on scheduling times and the LC algorithm had advantages on load balancing. The results showed the average response time of MIPSM to be similar to the Min–Min algorithm when $K < N$. MIPSM gradually shows its advantages when $K > N$. For resource utilization, MIPSM performed slightly better than the LC algorithm when $K < N$. However, MIPSM had more advantages when $K > N$. The deadline violation rate of MIPSM showed the best performance out of the three methods because MIPSM not only considered the state of tasks but also that of resources. Consequently, it balanced loads and improved resource utilization, especially for large numbers of tasks, demonstrating a greater advantage than other methods.

## References

[1] C. Zhu, V. C. M. Leung, X. Hu, L. Shu, and L. T. Yang, "A review of key issues that concern the feasibility of mobile cloud computing," in *Proc. IEEE Int. Conf. Cyber, Phys., Soc. Comput. (CPSCom)*, 2013, pp. 769–776.

[2] A. Beloglazov and R. Buyya, "Managing overloaded hosts for dynamic consolidation of virtual machines in cloud data centers under quality of service constraints," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 7, pp. 1366–1379, Jul. 2013.

[3] Z. Xiao, W. Song, and Q. Chen, "Dynamic resource allocation using virtual machines for cloud computing environment," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 6, pp. 1107–1116, Jun. 2013.

[4] J. Luo, L. Rao, and X. Liu, "Dynamic resource allocation using virtual machines for cloud computing environment," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 3, pp. 775–784, Mar. 2014.

[5] J. Cao, K. Li, and I. Stojmenovic, "Optimal power allocation and load distribution for multiple heterogeneous multicore server processors across clouds and data centers," *IEEE Trans. Comput.*, vol. 63, no. 1, pp. 45–58, Jan. 2014.

[6] B. Prabavathy, K. Priya, and C. Babu, "A load balancing algorithm for private cloud storage," in *Proc. 4th IEEE Int. Conf. Comput. Commun. Netw. Technol. (ICCCNT)*, 2013, pp. 1–6.

[7] R. A. M. Razali, R. A. Rahman, N. Zaini, and M. Samad, "Virtual machine migration implementation in load balancing for Cloud computing," in *Proc. 5th Int. Conf. Intell. Adv. Syst. (ICIAS)*, Kuala Lumpur, Malaysia, Jun. 3–5, 2014, pp. 1–4.

[8] L. Yang *et al.* "Real-time tasks oriented energy-aware scheduling in virtualized clouds," *IEEE Trans. Cloud Comput.*, vol. 2, no. 2, pp. 168–180, Feb. 2014.

[9] C. Tsai *et al.* "A hyper-heuristic scheduling algorithm for cloud," *IEEE Trans. Cloud Comput.*, vol. 2, no. 2, pp. 236–250, Feb. 2014.

[10] P. P. Hung, T. A. Bui, and E. N. Huh, "A new approach for task scheduling optimization in mobile cloud computing," in *Frontier and Innovation in Future Computing and Communications*, New York, NY, USA: Springer, 2014, pp. 211–220.

[11] T. Xiao *et al.* "A novel security-driven scheduling algorithm for precedence-constrained tasks in heterogeneous distributed systems," *IEEE Trans. Comput.*, vol. 60, no. 7, pp. 1017–1029, Jul. 2011.

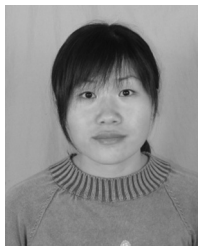
[12] Z. Tang *et al.* "A self-adaptive scheduling algorithm for reduce start time," *Future Gener. Comput. Syst.*, vol. 43, no. 6, pp. 51–60, Jun. 2015.

[13] S. Di, C. Wang, and F. Cappello, "Adaptive algorithm for minimizing cloud task length with prediction errors," *IEEE Trans. Cloud Comput.*, vol. 2, no. 2, pp. 194–206, Feb. 2014.

[14] Y. Wang and W. Shi, "Budget-driven scheduling algorithms for batches of MapReduce jobs in heterogeneous clouds," *IEEE Trans. Cloud Comput.*, vol. 2, no. 3, pp. 306–319, Mar. 2014.

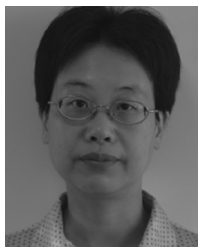
[15] Q. Zhang, M. F. Zhani, R. Boutaba, and J. H. L. Hellerstein, "HARMONY: Dynamic heterogeneity? Aware resource provisioning in the cloud," in *Proc. 33rd IEEE Int. Conf. Distrib. Comput. Syst.*, 2013, pp. 511–519.

[16] B. Xu, C. Zhao, E. Hu, and B. Hu, "Job scheduling algorithm based on Berger model in cloud environment," *Adv. Eng. Softw.*, vol. 42, no. 4, pp. 419–425, Apr. 2011.
[17] Q. Zhang, J. L. Hellerstein, and R. Boutaba, "Characterizing task usage shapes in Google's compute clusters," in *Proc. Large Scale Distrib. Syst. Middleware Workshop (LADIS)*, 2011, pp. 1–6.
[18] I. S. Moreno, P. Garraghan, P. Townend, and J. Xu, "An approach for characterizing workloads in Google cloud to derive realistic resource utilization models," in *Proc. 7th IEEE Int. Symp. Serv.-Oriented Syst. Eng.*, 2013, pp. 49–60.
[19] A. K. Mishra, J. L. Hellerstein, W. Cirne, and C. R. Das, "Towards characterizing cloud backend workloads: insights from Google compute clusters," in *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 37, no. 4, pp. 34–41, Mar. 2010.
[20] A. Khan, X. Yan, S. Tao, and N. Anerousis, "Workload characterization and prediction in the cloud: A multiple time series approach," in *Proc. IEEE Netw. Oper. Manage. Symp. (NOMS)*, 2012, pp. 1287–1294.
[21] M. Schwarzkopf and A. Konwinski, M. Abd-El-Malek, and J. Wilkes, "Omega: flexible, scalable schedulers for large compute clusters," in *Proc. 8th ACM Eur. Conf. Comput. Syst.*, 2013, pp. 351–364.
[22] B. Tripathy, S. Dash, and S. K. Padhy, "Dynamic task scheduling using a directed neural network," *J. Parallel Distrib. Comput.*, vol. 75, no. 1, pp. 101–106, Jan. 2015.
[23] H. Topcuoglu, S. Hariri, and M. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 3, pp. 260–274, Mar. 2002.
[24] C. C. Li and K. Wang, "An SLA-aware load balancing scheme for cloud datacenters," in *Proc. IEEE Int. Conf. Inf. Netw. (ICOIN)*, 2014, pp. 56–63.
[25] M. Mesbahi, A. M. Rahmani, and A. T. Chronopoulos, "Cloud light weight: A new solution for load balancing in cloud computing," in *Proc. Int. Conf. Data Sci. Eng. (ICDSE)*, 2014, pp. 44–50.
[26] Y. F. Wen and C. L. Chang, "Load balancing job assignment for cluster-based cloud computing," in *Proc. 6th IEEE Int. Conf. Ubiq. Future Netw. (ICUFN)*, 2014, pp. 199–204.
[27] T. Wang *et al.* "Load balancing task scheduling based on genetic algorithm in cloud computing," in *Proc. 12th IEEE Int. Conf. Depend., Auton. Secure Comput. (DASC)*, 2014, pp. 146–152.
[28] H. Chen *et al.* "User-priority guided Min-Min scheduling algorithm for load balancing in cloud computing," in *Proc. Natl. Conf. Parallel Comput. Technol. (PARCOMPTECH)*, 2013, pp. 1–8.
[29] V. Behal and A. Kumar, "Cloud computing: Performance analysis of load balancing algorithms in cloud heterogeneous environment," in *Proc. 5th Int. Conf. Confluence Next Gener. Inf. Technol. Summit (Confluence)*, 2014, pp. 200–205.
[30] K. A. Nuaimi *et al.* "A survey of load balancing in cloud computing: Challenges and algorithms," in *Proc. 2nd Symp. Netw. Cloud Comput. Appl. (NCCA)*, 2012, pp. 137–142.
[31] M. A. Rodriguez and R. Buyya, "Deadline based resource provisioning and scheduling algorithm for scientific workflows on clouds," *IEEE Trans. Cloud Comput.*, vol. 2, no. 2, pp. 222–235, Feb. 2014.
[32] M. Hu and B. Veeravalli, "Dynamic scheduling of hybrid real-time tasks on clusters," *IEEE Trans. Comput.*, vol. 63, no. 12, pp. 2988–2997, Dec. 2014.
[33] K. Li and X. Tang, "Energy-efficient stochastic task scheduling on heterogeneous computing systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 11, pp. 2967–2876, Nov. 2014.
[34] P. Agrawal and S. Rao, "Energy-aware scheduling of distributed systems," *IEEE Trans. Autom. Sci. Eng.*, vol. 11, no. 4, pp. 1163–1175, Apr. 2014.
[35] J. Mei, K. Li, and K. Li, "Energy-aware task scheduling in heterogeneous computing environments," *Cluster Comput.*, vol. 12, no. 2, pp. 537–550, Feb. 2014.
[36] R. Duan, R. Prodan, and X. Li, "Multi-objective game theoretic scheduling of bag-of-tasks workflows on hybrid clouds," *IEEE Trans. Cloud Comput.*, vol. 2, no. 1, pp. 29–42, Jan. 2014.
[37] Z. Liu *et al.* "Resource preprocessing and optimal task scheduling in cloud computing environments," *Concurrency Comput. Pract. Exp.*, vol. 31, no. 2, pp. 1–22, Feb. 2014.
[38] Q. Zhu and G. Agrawal, "Resource provisioning with budget constraints for adaptive applications in cloud environments," *IEEE Trans. Serv. Comput.*, vol. 5, no. 4, pp. 497–511, May 2011.
[39] R. N. Calheiros, R. Ranjan, A. Beloglazov, A. F. Cesar, R. De, and R. Buyya, "CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Softw. Pract. Exper.*, vol. 41, no. 1, pp. 23–50, 2011.
[40] G. Liu, J. Li, and J. Xu, "An improved min-min algorithm in cloud computing," in *Proc. Int. Conf. Modern Comput. Sci. Appl.*, 2013, pp. 47–52.
[41] C. Zhu, X. Li, V. C. M. Leung, X. Hu, and L. T. Yang, "Job scheduling for cloud computing integrated with wireless sensor network," in *Proc. 6th IEEE Int. Conf. Cloud Comput. Technol. Sci. (CloudCom)*, 2014, pp. 62–69.
[42] Z. Yang, L. Huang, and M. Xiao, "Flow-based transmission scheduling in constrained delay tolerant networks," *J. Comput.*, vol. 7, no. 1, pp. 179–186, 2012.
[43] Z. Cao, S. Dong, B. Wang, and L. Zuo, "Workload analysis and modeling of high performance computing trace of biological gene sequencing," *J. Softw.*, vol. 25, no. S2, pp. 90–100, 2014.
[44] G. Han, Y. Dong, H. Guo, L. Shu, and D. Wu, "Cross-layer optimized routing in wireless sensor networks with duty-cycle and energy harvesting," *Wireless Commun. Mobile Comput.*, vol. 15, no. 16, pp. 1957–1981, 2015.
[45] G. Han, W. Que, G. Jia, and L. Shu, "An efficient virtual machine consolidation scheme for multimedia cloud computing," *JSensors*, vol. 16, no. 2, pp. 1–17, 2016.



**Liyun Zuo** received the B.S. degree in engineering and management from Zhengzhou University, Zhengzhou, China, in 2003, and the M.S. degree in computer science from Huazhong University of Science and Technology (HUST), Wuhan, China, in 2006. Currently, she is pursuing the Ph.D. degree in computer science and engineering at South China University of Technology (SCUT), Guangzhou, China, from September 2013.

She is an Associate Professor with Guangdong University of Petrochemical Technology, Maoming, China. Her research interests include cloud computing, resource evaluation, and scheduling optimization.

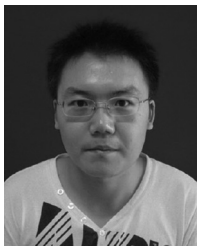


**Shoubin Dong** received the Ph.D. degree in electronic engineering from the University of Science and Technology of China (USTC), in 1994.

She is a Professor with the School of Computer Science and Engineering, South China University of Technology (SCUT), Guangzhou, China. She was a Visiting Scholar at the School of Computer Science, Carnegie Mellon University (CMU), Pittsburgh, PA, USA, from 2001 to 2002. She is currently the Deputy Director of Communication and Computer Network Laboratory (CCNL), Guangdong Province, China. Her research interests include high-performance computing, big data processing, and next generation Internet.

**Lei Shu** (M'07–SM'16) received the Ph.D. degree in computer engineering from the National University of Ireland, Galway, Ireland, in 2010. Until March 2012, he was a Specially Assigned Researcher with the Department of Multimedia Engineering, Graduate School of Information Science and Technology, Osaka University, Suita, Japan. Since October 2012, he has been a Full Professor with Guangdong University of Petrochemical Technology, Maoming, China. Since 2013, he has been serving as a Ph.D. Supervisor at Dalian University of Technology, Dalian, China, and as a Master Supervisor Beijing University of Posts and Telecommunications, Beijing, China. Meanwhile, he is also working as the Vice-Director of the Guangdong Provincial Key Laboratory of Petrochemical Equipment Fault Diagnosis, Maoming, China. He is the Founder of Industrial Security and Wireless Sensor Networks Laboratory. He has authored more than 200 papers in related conferences, journals, and books. His current H-index is 17. His research interests include wireless sensor networks, multimedia communication, middleware, security, and fault diagnosis.

Dr. Shu is a member of European Alliance for Innovation (EAI), and Association for Computing Machinery (ACM). He is serving as the Editor in Chief for *EAI Endorsed Transactions on Industrial Networks and Intelligent Systems*, and an Associate Editor for a number of famous international journals. He served as a Co-Chair for more than 50 various for international conferences/workshops, e.g., IEEE International Wireless Communications & Mobile Computing Conference (IWCMC), IEEE International Conference on Communications (ICC), IEEE Symposium on Computers and Communications (ISCC), IEEE International Conference on Computing, Networking, and Communications (ICNC), Chinacom, especially the Symposium Co-Chair for IWCMC 2012, ICC 2012, the General Chair for Chinacom 2014, Qshine 2015, the Steering Chair for InisCom 2015; TPC members of more than 150 conferences, e.g., International Conference on Distributed Computing in Sensor Systems (DCOSS), IEEE International Conference on Mobile Ad hoc and Sensor Systems (MASS), ICC, Globecom, International Conference on Computer Communication and Networks (ICCCN), IEEE Wireless Communications and Networking Conference (WCNC), ISCC. He was the recipient of the Globecom 2010 and ICC 2013 Best Paper Award.



**Chunsheng Zhu** (S'12) received the B.E. degree in network engineering from Dalian University of Technology, Dalian, China, in 2010, and the M.Sc. degree in computer science from St. Francis Xavier University, Antigonish, NS, Canada, in 2012. Currently, he is pursuing the Ph.D. degree in electrical and computer engineering at the University of British Columbia, Vancouver, BC, Canada, from September 2012.

He has more than 50 published or accepted papers by refereed international journals (e.g., the IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS, the IEEE TRANSACTIONS ON COMPUTERS, IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY, IEEE TRANSACTIONS ON EMERGING TOPICS IN COMPUTING, IEEE SYSTEMS JOURNAL, IEEE ACCESS) and conferences (e.g., IEEE Globecom, IEEE ICC). His research interests include wireless sensor networks and cloud computing.

**Guangjie Han** (S'03–M'05) received the Ph.D. degree in computer science from the Northeastern University, Shenyang, China, in 2004.

Currently, he is a Professor with the Department of Information and Communication System, Hohai University, Nanjing, China. He was also a Visiting Research Scholar at Osaka University, Suita, Japan, from 2010 to 2011. He was a Post-Doctoral Researcher of Computer Science with Chonnam National University, Gwangju, Korea, in February 2008. He worked with ZTE Company, Shenzhen, China, from 2004 to 2006, where he held the position of Product Manager. He has authored more than 100 papers in related international conferences and journals. He holds 37 patents. He has served as a reviewer of more than 50 journals. His research interests include sensor networks, computer communications, mobile cloud computing, multimedia communication, and security.

Dr. Han is a member of ACM. He has served in the editorial board of up to seven international journals, including *Journal of Internet Technology* and *KSII Transactions on Internet and Information Systems*. He has served as a Co-Chair for more than 20 international conferences/workshops; a TPC member of more than 50 conferences.